Co-occurrence-based recommendations with Mahout, Scala & Spark

Christoph Boden (TU Berlin)
Budapest DW FORUM, Big Data Day
06/04/2014

[filling in for: Sebastian Schelter, @sscdotopen]

Mahout: Past & Future

Apache Mahout: History

- library for **scalable machine learning (ML)**
- **started** six years ago **as ML on MapReduce**
- focus on popular ML problems and algorithms
  - **Collaborative Filtering**
    - **(Item-based, User-based, Matrix Factorization)**
  - Classification
    - (Naive Bayes, Logistic Regression, Random Forest)
  - Clustering
    - (k-Means, Streaming k-Means)
  - Dimensionality Reduction
    - (Lanczos, Stochastic SVD)

  + in-memory math library (fork of Colt)
- **large userbase** (e.g. Adobe, AOL, Accenture, Foursquare, Mendeley, Researchgate, Twitter)

Apache Mahout: Problems

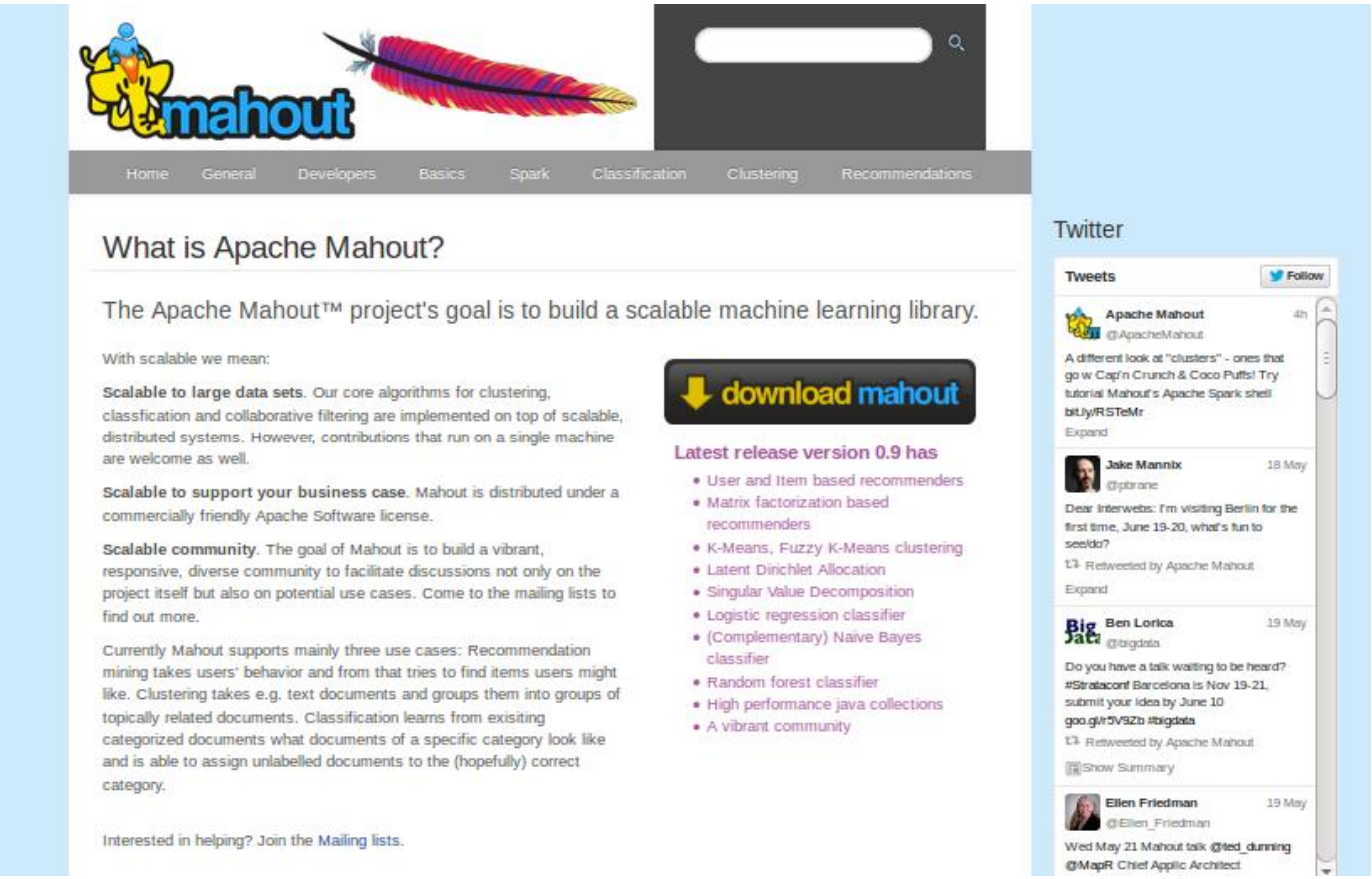
- **Organisational**
  - Onerous to provide support and documentation for scalable ML (convey technical and theoretical background at the same time)
  - partly failed to compensate for developers that left the project
  - barrier for contributions too low

- **Technical**
  - MapReduce not well suited for ML
    - slow execution, especially for iterations
    - constrained programming model makes code hard to write, read and adjust
  - different quality of implementations
  - lack of unified preprocessing machinery

Apache Mahout: Future

- **Narrowing of focus**
  - removed a large number of rarely used or barely maintained algorithm implementations
- **Redesign of website**
  - in the process of reworking documentation for implemented algorithms
- **Abandonment of MapReduce**
  - will reject new MapReduce implementations
  - widely used „legacy“ implementations will be maintained
  - Hadoop 2 compatibility for „legacy“ implementations
- **„Reboot“**
  - usage of **modern parallel processing systems** which offer richer progamming model & more efficient execution
  - **DSL for linear algebraic operations** as foundation for new implemenations
  - **automatic optimization and parallelization** of programs written in this DSL

Shiny new website



Committers working on upcoming release

- **66 contributions** since last release (february)
- **participation of 12 committers**

Gokhan Capan (Dilisim, Turkey)
Isabel Drost-Fromm (ElasticSearch, Germany)
Suneel Marthi (Intel, US)
David Weiss (Carrot Search, Poland)
Frank Scholten (Trifork, Netherlands)
Andrew Musselman (Accenture, US)
Sebastian Schelter (TU Berlin, Germany)
Sean Owen (Cloudera, UK)
Pat Ferrel (Finderbots, US)
Ted Dunning (MapR Technologies, US)
Dmitriy Lyubimov (Agil One, US)
Stevo Slavic (Hybris Software, Germany)

50% of work on upcoming release done by external people

- **external contributions for 32** out of those 66 contributions from **21 different people**

Sergey Svinarchuk
till.rohrmann
Manoj Awasthi
Pat Ferrel
Adam Ilardi
Kevin Moulart
Maciej Mazur

Steve Cook
Pavan Kumar
Gaurav Misra
Avi Shinnar
Maxim Arap
Jian Wang
Saleem Ansari

Andrew Palumbo
Nick Martin
Terry Blankers
Dodi Hakim
Johannes Schulte
Sergey
Yexi Jiang

Reboot

(Scala & Spark-Bindings)

Requirements for an ideal ML environment

1. **R/Matlab-like semantics**
   - type system that covers **(a) linear algebra**, **(b) statistics** and **(c) data frames**

2. **Modern programming language qualities**
   - functional programming
   - object oriented programming
   - sensible byte code Performance
   - scriptable and Interactive

3. **Scalability**
   - automatic distribution and parallelization with sensible performance

4. **Collection of off-the-shelf building blocks and algorithms**

5. **Visualization**

Requirements for an ideal ML environment

1. **R/Matlab-like semantics**
   - type system that covers **(a) linear algebra**, **(b) statistics** and **(c) data frames**

2. **Modern programming language qualities**
   - functional programming
   - object oriented programming
   - sensible byte code Performance
   - scriptable and Interactive

3. **Scalability**
   - automatic distribution and parallelization with sensible performance

4. **Collection of off-the-shelf building blocks and algorithms**

5. **Visualization**

Requirements for an ideal ML environment

1. **R/Matlab-like semantics**
   - type system that covers **(a) linear algebra**, **(b) statistics** and **(c) data frames**

2. **Modern programming language qualities**
   - functional programming
   - object oriented programming
   - sensible byte code Performance
   - scriptable and Interactive

3. **Scalability**
   - automatic distribution and parallelization with sensible performance

4. **Collection of off-the-shelf building blocks and algorithms**

5. **Visualization**

Requirements for an ideal ML environment

1. **R/Matlab-like semantics**
   - type system that covers **(a) linear algebra**, **(b) statistics** and **(c) data frames**

2. **Modern programming language qualities**
   - functional programming
   - object oriented programming
   - sensible byte code Performance
   - scriptable and Interactive

3. **Scalability**
   - automatic distribution and parallelization with sensible performance

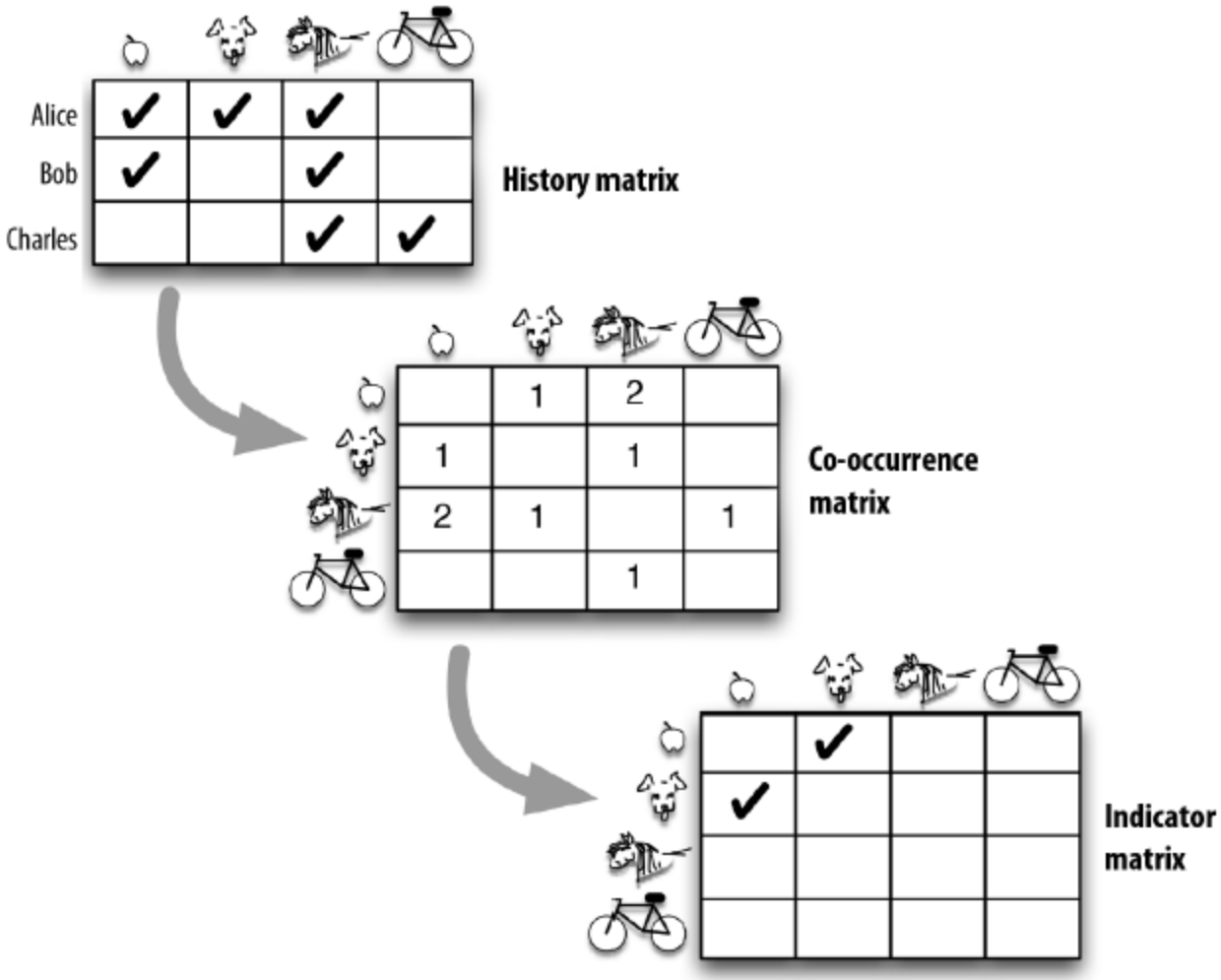
4. **Collection of off-the-shelf building blocks and algorithms**

5. ~~**Visualization**~~

Example

available for free at
http://www.mapr.com/practical-machine-learning

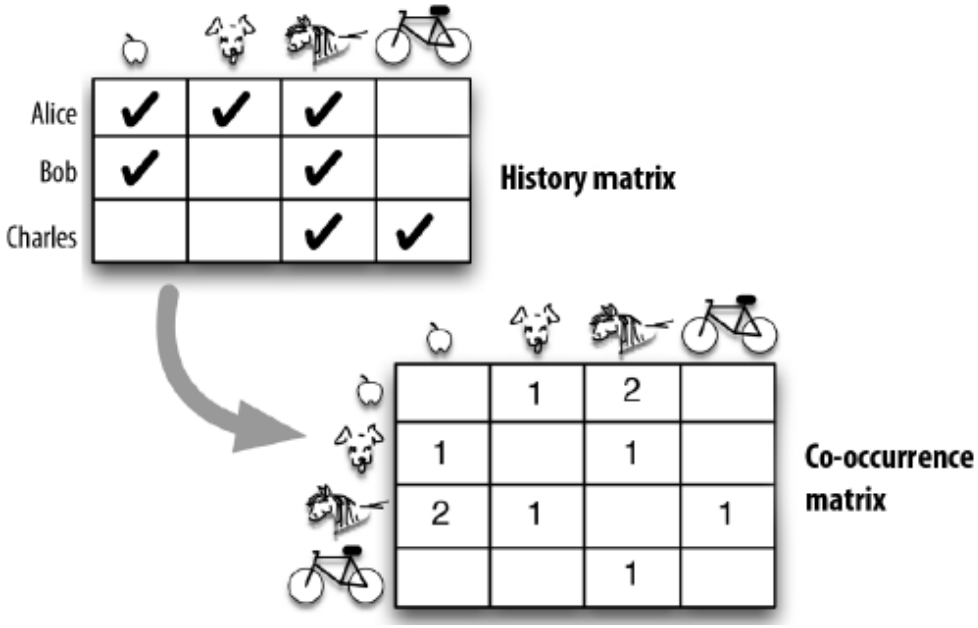
Cooccurrence Analysis



History matrix

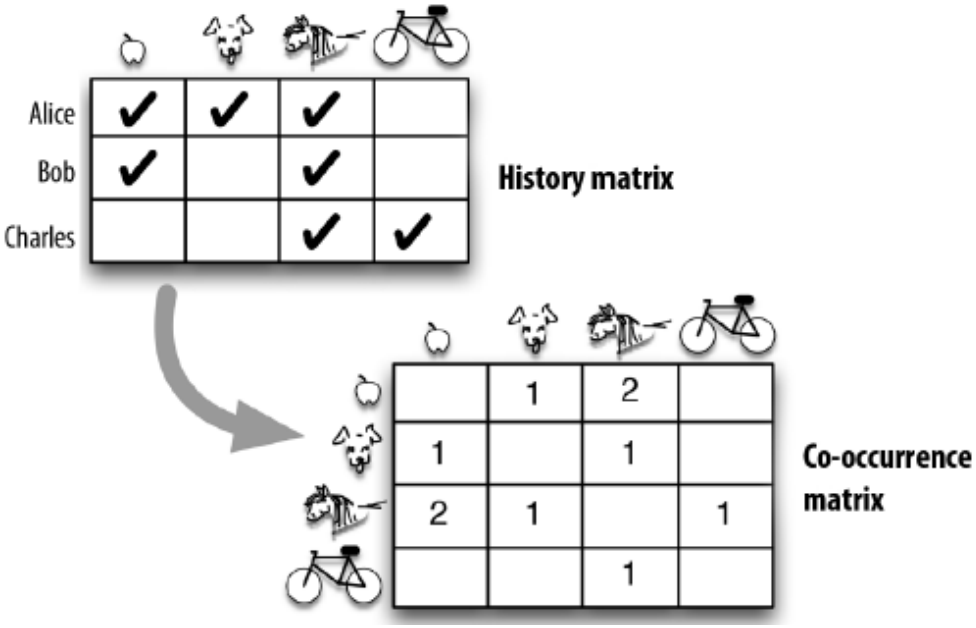
```
// real usecase: load from DFS
// val A = drmFromHDFS(...)


// our toy example
val A = drmParallelize(dense(
        (1, 1, 1, 0),    // Alice
        (1, 0, 1, 0),    // Bob
        (0, 0, 1, 1)),   // Charles
        numPartitions = 2)
```

How often do items co-occur?

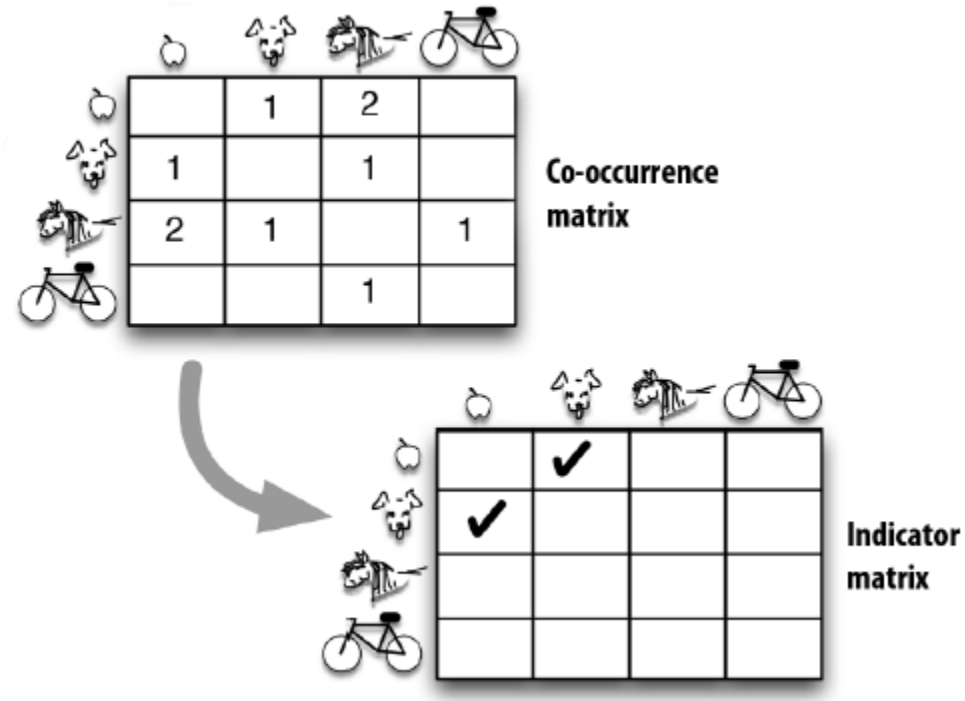


How often do items co-occur?

```
// compute co-occurrence matrix
val C = A.t %*% A
```

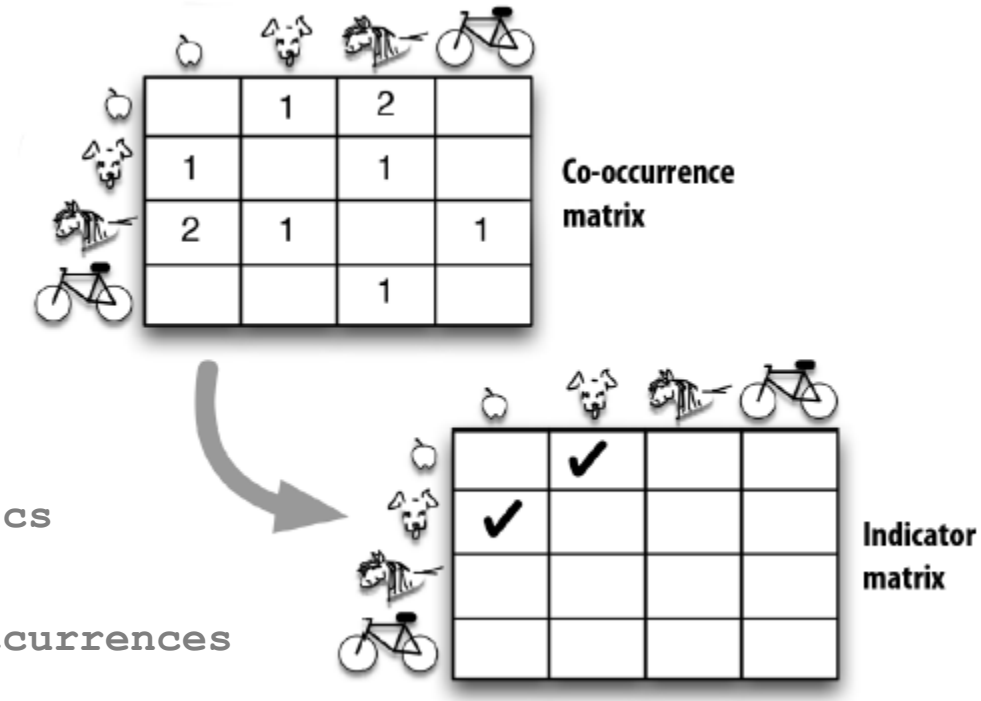


Which cooccurences are interesting?



Which cooccurences are interesting?

```
// compute some statistics
val interactionsPerItem =
   drmBroadcast(A.colSums)

// convert to indicator matrix
val I = C.mapBlock() {
   // compute LLR scores from
   // cooccurrences and statistics
   ...
   // only keep interesting cooccurrences
   ...
}

// save indicator matrix
I.writeDrm(...);
```



Cooccurrence Analysis prototype available

- **MAHOUT-1464** provides full-fledged cooccurrence analysis protoype
  - applies selective downsampling to make computation tractable
  - **support for cross-recommendations in datasets with multiple interaction types**, e.g.
    - *"people who watch this video also watch those videos"*
    - *"people who enter this search query watch those videos"*
  - code to run this on the Movielens and Epinions datasets

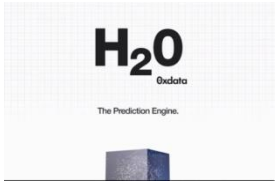
- future plan: easy **indexing of indicator matrix with Apache Solr** to allow for **search-as-recommendation** deployments
  - prototype for MR code already existing at https://github.com/pferrel/solr-recommender
  - integration is in the works

Under the covers

Underlying systems

- currently: runtime based on **Apache Spark**
  - fast and expressive cluster computing system
  - general computation graphs, in-memory primitives, rich API, interactive shell

- potentially supported in the future:
  - **Stratosphere**
  - **H20**



Runtime & Optimization

- Execution is deferred, user composes logical operators

```
val C = A.t %*% A
```

- Computational actions implicitly trigger optimization (= selection of physical plan) and execution

```
I.writeDrm(path);
```

```
val inMemV =(U %*% M).collect
```

- Optimization factors: size of operands, orientation of operands, partitioning, sharing of computational paths

- e. g.: matrix multiplication:
  - 5 physical operators for drmA %*% drmB
  - 2 operators for drmA %*% inMemA
  - 1 operator for drm A %*% x
  - 1 operator for x %*% drmA

Optimization Example

- Computation of A^TA in example

```
val C = A.t %*% A
```

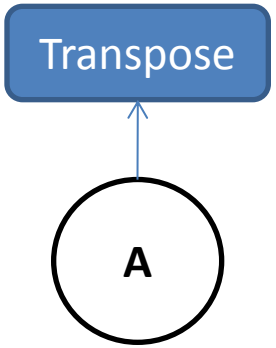
Optimization Example

- Computation of A^TA in example

```
val C = A.t %*% A
```

- **Naïve execution**

  1st pass: transpose A
  (requires repartitioning of A)



Optimization Example

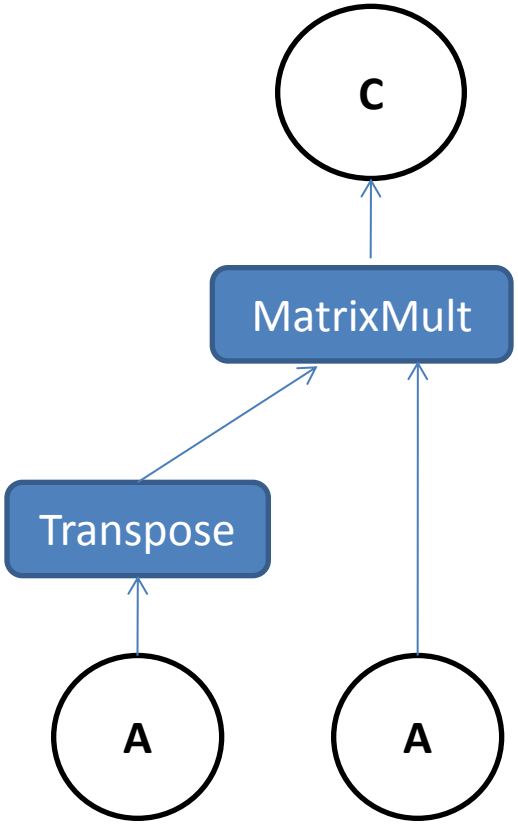
- Computation of A^TA in example

```
val C = A.t %*% A
```

- Naïve execution

  1st pass: transpose A
  (requires repartitioning of A)

  2nd pass: multiply result with A
  (expensive, potentially requires repartitioning again)



Optimization Example

- Computation of A^TA in example

```
val C = A.t %*% A
```

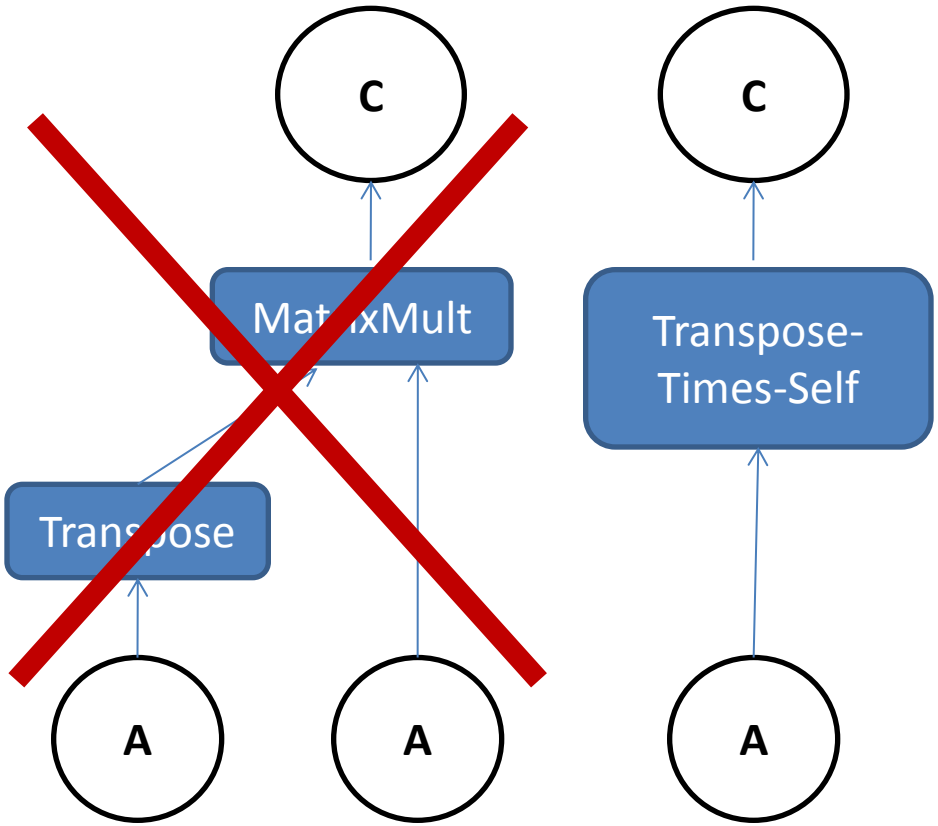
- Naïve execution

  1st pass: transpose A (requires repartitioning of A)

  2nd pass: multiply result with A (expensive, potentially requires repartitioning again)

- **Logical optimization**

  Optimizer rewrites plan to use specialized logical operator for *Transpose-Times-Self* matrix multiplication



Tranpose-Times-Self

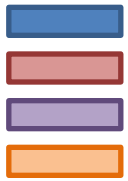
- Mahout computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

$$\boldsymbol{A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{\ T}}$$

Tranpose-Times-Self

- Mahout computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{\ T}$$

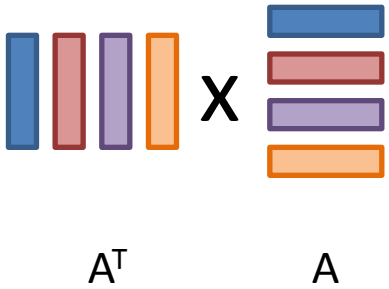


A

Tranpose-Times-Self

- Mahout computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

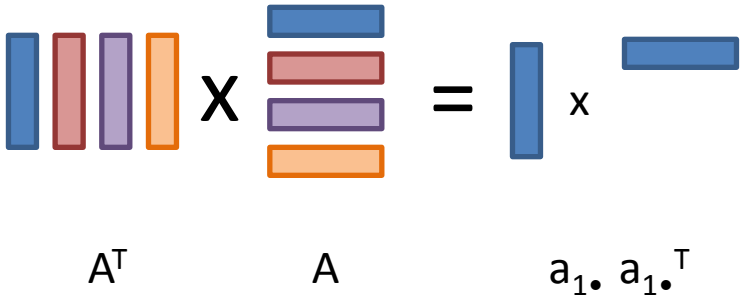
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{\ T}$$



Tranpose-Times-Self

- Mahout computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

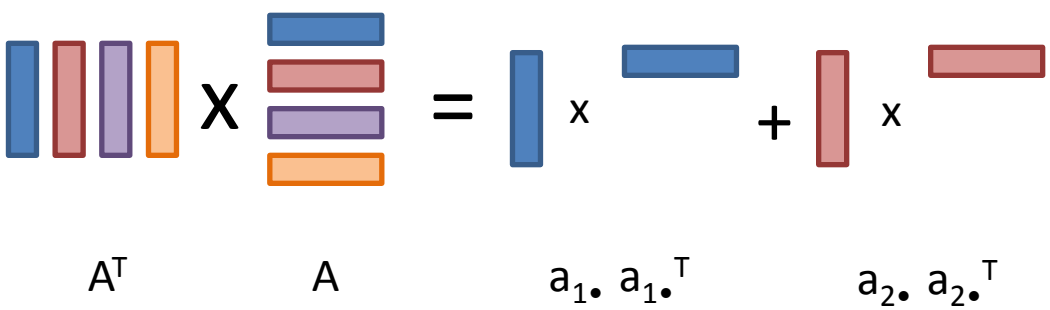
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{\ T}$$



Tranpose-Times-Self

- Mahout computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

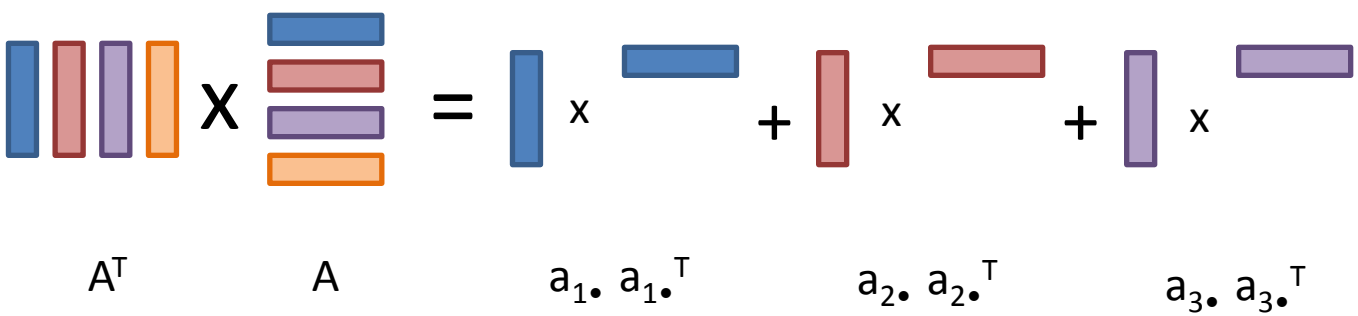
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{\ T}$$



Tranpose-Times-Self

- Mahout computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

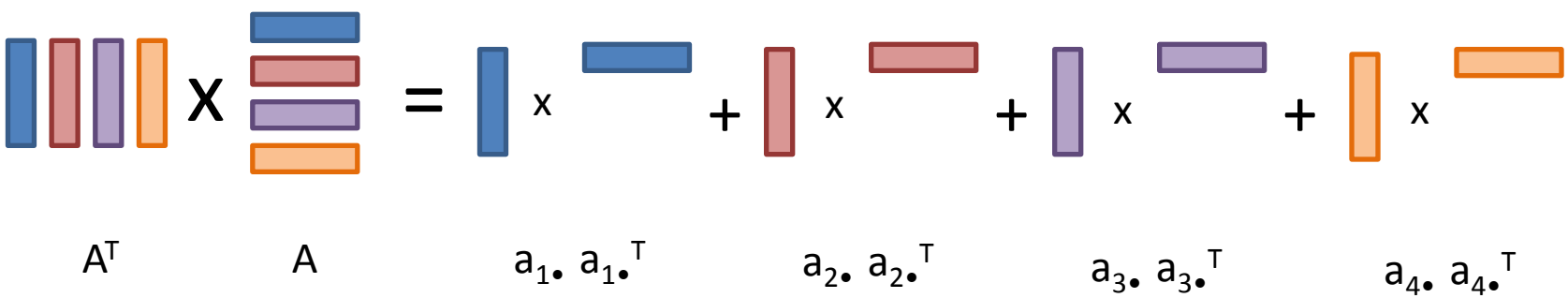
$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{\ T}$$



Tranpose-Times-Self

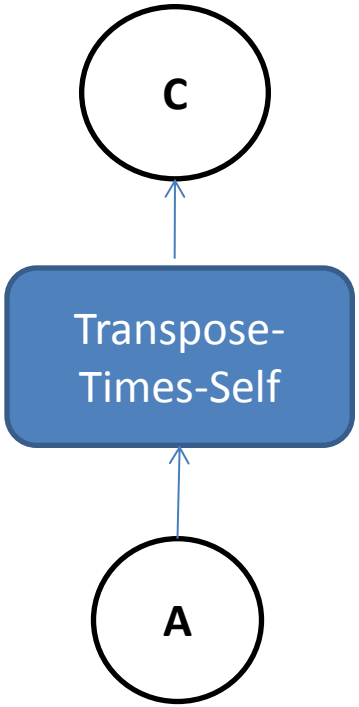
- Mahout computes A^TA via **row-outer-product** formulation
  - executes in a single pass over row-partitioned A

$$A^T A = \sum_{i=0}^{m} a_{i\bullet} a_{i\bullet}^{\ T}$$



Physical operators for Transpose-Times-Self

- Two physical operators (concrete implementations) available for *Transpose-Times-Self* operation
  - standard operator ***AtA***
  - operator ***AtA_slim,*** specialized implementation for tall & skinny matrices
- Optimizer must choose
  - currently: depends on user-defined threshold for number of columns
  - ideally: cost based decision, dependent on estimates of intermediate result sizes



- Started out 6 years ago as a **scalable machine learning library on MapReduce**
- core algorithms for **clustering**, **classification** and **collaborative filtering**
- will **reject new MapReduce implementations**
- widely used **„legacy“ implementations will be maintained**

- future implementations on top of a **DSL for linear algebraic operations** based on **Scala**

- Programs written in this DSL are **automatically optimized** and **executed in parallel** on **Apache Spark** (potentially others: Stratosphere, H20, …)

Playing with Mahout's Spark Shell

This tutorial will show you how to play with Mahout's scala DSL for linear algebra and its Spark shell. **Please keep in mind that this code is still in a very early experimental stage**.

Intro

We'll use an excerpt of a publicly available dataset about cereals. The dataset tells the protein, fat, carbohydrate and sugars (in milligrams) contained in a set of cereals, as well as a customer rating for the cereals. Our aim for this example is to fit a linear model which infers the customer rating from the ingredients.

Name	protein	fat	carbo	sugars	rating
Apple Cinnamon Cheerios	2	2	10.5	10	29.509541
Cap'n'Crunch	1	2	12	12	18.042851
Cocoa Puffs	1	1	12	13	22.736446
Froot Loops	2	1	11	13	32.207582
Honey Graham Ohs	1	2	12	11	21.871292
Wheaties Honey Gold	2	1	16	8	36.187559
Cheerios	6	2	17	1	50.764999
Clusters	3	2	13	7	40.400208

Thank you. Questions?

Overview of Mahout's Scala & Spark Bindings:

http://s.apache.org/mahout-spark

Tutorial on playing with Mahout's Spark shell

http://s.apache.org/mahout-spark-shell

Credits:
Design & implementation of the Scala & Spark Bindings and parts of this slideset by Dmitriy Lyubimov ***@dlieuOfTwit***

Slides by Sebastian Schelter ***@sscdotopen***

Backup

Data Types

- Scalar real values

```
val x = 2.367
```

- In-memory vectors
  - dense
  - 2 types of sparse

```
val v = dvec(1, 0, 5)

val w =
  svec((0 -> 1)::(2 -> 5)::Nil)
```

- In-memory matrices
  - sparse and dense
  - a number of specialized matrices

```
val A = dense((1, 0, 5),
              (2, 1, 4),
              (4, 3, 1))
```

- Distributed Row Matrices (DRM)
  - huge matrix, partitioned by rows
  - lives in the main memory of the cluster
  - provides small set of parallelized operations
  - lazily evaluated operation execution

```
val drmA = drmFromHDFS(...)
```

Features (1)

- Matrix, vector, scalar operators: in-memory, out-of-core

```
drmA %*% drmB
A %*% x
A.t %*% drmB
A * B
```

- Slicing operators

```
A(5 until 20, 3 until 40)
A(5, ::); A(5, 5)
x(a to b)
```

- Assignments (in-memory only)

```
A(5, ::) := x
A *= B
A -=: B; 1 /:= x
```

- Vector-specific

```
x dot y; x cross y
```

- Summaries

```
A.nrow; x.length;
A.colSums; B.rowMeans
x.sum; A.norm
```

Features (2)

- solving linear systems

```
val x = solve(A, b)
```

- in-memory decompositions

```
val (inMemQ, inMemR) = qr(inMemM)
val ch = chol(inMemM)
val (inMemV, d) = eigen(inMemM)
val (inMemU, inMemV, s) = svd(inMemM)
```

- out-of-core decompositions

```
val (drmQ, inMemR) = thinQR(drmA)
val (drmU, drmV, s) =
  dssvd(drmA, k = 50, q = 1)
```

- caching of DRMs

```
val drmA_cached = drmA.checkpoint()
drmA_cached.uncache()
```

- custom modification of DRMs

```
val modifiedA = drmA.mapBlock(){
  case (keys, block) => …
}
```